

lib/main/power_unittest.ath

```

1 load "power"
2
3 -----
4 define M1 := no-renaming
5
6 assert (theory-axioms Monoid.theory)
7
8 (!prove-property Monoid.Power.right-plus M1 Monoid.theory)
9
10 (!prove-property Monoid.Power.left-neutral M1 Monoid.theory)
11
12 (!prove-property Monoid.Power.right-one M1 Monoid.theory)
13 (!prove-property Monoid.Power.right-two M1 Monoid.theory)
14 (!prove-property Monoid.Power.right-times M1 Monoid.theory)
15
16 assert (get-property Abelian-Monoid.commutative M1 Abelian-Monoid.theory)
17
18 (!prove-property Abelian-Monoid.Power-left-times M1 Abelian-Monoid.theory)
19
20 -----
21 load "nat-power"
22
23 define M1 := (renaming |{Monoid.** := N.**, Monoid.+ := N.*,
24                 Monoid.<0> := N.one}|)
25
26 define [n x] := [?n:N ?x:'T]
27
28 (!claim (forall n x . (N.** x (S n) = x N.* (N.** x n)))
29
30 (!prove-property Monoid.Power.right-plus M1 Monoid.theory)
31
32 (!prove-property Monoid.Power.left-neutral M1 Monoid.theory)
33
34 (!prove-property Monoid.Power.right-one M1 Monoid.theory)
35
36 (!prove-property Monoid.Power.right-two M1 Monoid.theory)
37
38 (!prove-property Monoid.Power.right-times M1 Monoid.theory)
39
40 (!prove-property Abelian-Monoid.Power-left-times M1 Abelian-Monoid.theory)
41
42 -----
43 declare **: (T) [T N] -> T [400]
44
45 define M1 := (renaming |{Monoid.** := **}|)
46
47 assert (M1 (theory-axioms MM.theory))
48
49 (!prove-property Monoid.Power.right-plus M1 MM.theory)
50
51 (!prove-property Monoid.Power.left-neutral M1 MM.theory)
52
53 (!prove-property Monoid.Power.right-one M1 MM.theory)
54
55 (!prove-property Monoid.Power.right-two M1 MM.theory)
56
57 (!prove-property Monoid.Power.right-times M1 MM.theory)
58
59 (assert (get-property Abelian-Monoid.commutative M1 MAM.theory))
60
61 (!prove-property Abelian-Monoid.Power-left-times M1 MAM.theory)
62
63 -----
64 load "list-of.ath"
65
66 declare Join*: (T) [(List T) N] -> (List T)
67

```

```

68 define M1 := (renaming |{Monoid.+* := Join*, Monoid.+ := List.join,
69                      Monoid.<0> := nil}|)
70
71 # Define Join* as an instance of **
72
73 assert (map lambda (P) (M1 P)
74          [Monoid.Power.right-zero Monoid.Power.right-nonzero])
75
76 (!prove-property Monoid.Power.right-plus M1 Monoid.theory)
77
78 (!prove-property Monoid.Power.left-neutral M1 Monoid.theory)
79
80 (!prove-property Monoid.Power.right-one M1 Monoid.theory)
81
82 (!prove-property Monoid.Power.right-two M1 Monoid.theory)
83
84 (!prove-property Monoid.Power.right-times M1 Monoid.theory)
85
86 # List.join isn't commutative, so we don't have Monoid.Power-left-times.
87
88 #-----
89 open Monoid
90
91 define L1 := (1 :: 2 :: nil)
92
93 define three := (S N.two)
94
95 #{set-debug-mode "rewriting"}
96
97 let {adapter := (renaming |{+* := Join*, + := List.join, <0> := nil}|);
98      [get prove chain chain-> chain<-] := (proof-tools adapter Monoid.theory);
99      [+ <0> +*] := (adapter [+ <0> +*]);
100     _ := (!prove Power.right-two)}
101 let {_ := (!claim Power.right-nonzero)}
102     (!chain [(L1 +* three)
103             --> (L1 + (L1 +* N.two))      [Power.right-nonzero]
104             --> (L1 + L1 + L1)           [Power.right-two]])
105
106 (eval (L1 List.join L1 List.join L1))

```